**B6x开发板说明**

**目录**

[内容简介 1](#_Toc17667)

[第1篇 硬件手册 2](#_Toc7477)

[第1章 开发板简介 3](#_Toc1886)

[1.1.1 QFN32开发板资源初探 3](#_Toc2070)

[1.1.2 QFN32开发板资源说明 4](#_Toc16902)

[第2篇 入门指南 6](#_Toc13021)

[第1章 软件开发环境搭建 7](#_Toc19346)

[2.1.1 MDK5简介 7](#_Toc10811)

[2.1.2 MDK5安装 8](#_Toc5269)

[2.1.3 Jlink驱动安装 8](#_Toc117)

[2.1.4 分散加载文件浅释 9](#_Toc16962)

[2.1.5 分散加载文件使用说明 12](#_Toc6661)

[第2章 下载与调试 14](#_Toc29421)

[2.2.1 程序下载 14](#_Toc24719)

[2.2.2 在线调试 15](#_Toc9158)

[第3篇 常见问题 20](#_Toc5690)

[3.1.1 找不到USB设备? 20](#_Toc10565)

[3.1.2 JLINK 能否给开发板供电？ 20](#_Toc5341)

[3.1.3 JLINK第一次烧录后无法再次烧录？ 20](#_Toc30750)

# 内容简介

本手册将由浅入深，带领大家学习B6x芯片的各个功能，为您开启全新的B6x之旅。

本手册总共分为三篇：

* 入门指南篇，主要介绍B6x常用开发软件的使用以及一些下载调试的技巧，并详细介绍了几个常用的文件（程序）；
* 硬件手册篇，主要介绍硬件开发手册，以及本手册硬件平台：QFN32开发板；
* 常见错误篇，主要收集一些开发中常见的问题，并给出一些解决方案。

本手册基于QFN32开发板（以下简称：开发板）做相关演示及实验，在开发板配套的资料里面，有详细原理图以及所有实例的完整代码，这些代码都有详细的注释，所有源码都经过我们严格测试，不会有任何错误，另外，编译源码有我们生成好的bin文件，大家只需要通过串口/仿真器下载到开发板即可看到实验现象，亲自体验实验过程。

本手册不仅非常适合广大学生和电子爱好者学习B6x，其大量的实验以及详细的解说，也是公司产品开发的不二参考。

我们都是经过检验之后的开发板才包装的,所以来到您手上的板子一般都是没问题的。这里不能说绝对没问题,因为板子在运输途中可能会有损坏。所以收到货后,请先检查开发板是否正常，检查方法如前面所述。

# 硬件手册

实践出真知，要想学好B6x，实验平台必不可少！本篇将详细介绍我们用来学习B6x的硬件平台：QFN32开发板，通过该篇的介绍，你将了解到我们的学习平台QFN32开发板的功能及特点。

为了让读者更好的使用QFN32开发板，本篇还介绍了开发板的一些使用注意事项，请读者在使用开发板的时候一定要注意。

本篇将分为如下两章：

1. 硬件设计；
2. 实验平台简介；

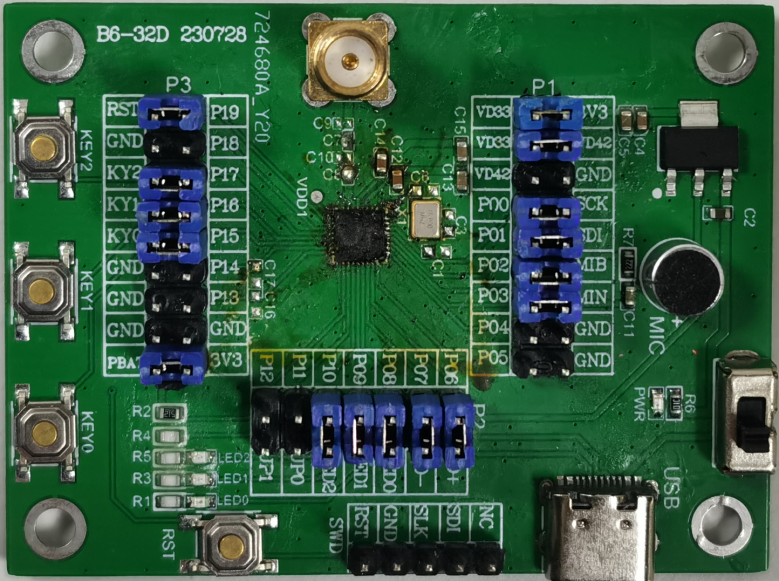
## 开发板简介

本章，主要向大家简要介绍我们的实验平台：QFN32开发板。通过本章的学习，你将对我们后面使用的实验平台有个大概了解，为后面的学习做铺垫。

本章将分为如下两节：

### QFN32开发板资源初探

这款QFN32开发板，是32位开发板，下面我们开始介绍QFN32开发板的资源图，如图 1‑1所示：



**引出**

**IO口**

**SWD**

**接口**

**复位**

**按钮**

**电源开关**

**电源**

**指示灯**

**麦克风**

**QFN32芯片**

**RF**

**天线**

**引出**

**IO口**

**引出IO口**

**3个按键**

**3个LED灯**

**晶振**

**16M**

**3.3V LDO**

**USB口**

图 1‑1 QFN32开发板资源图

从图 4‑37可以看出，QFN32开发板，资源十分丰富，并把QFN32的内部资源发挥到了极致，基本所有QFN32的内部资源，都可以在此开发板上验证。

开发板的外形尺寸为47mm\*63mm，QFN32开发板板载资源如下：

* + - MCU：QFN32，FLASH：256K，SRAM：32K；
    - 1个电源指示灯（PWR）
    - 3个状态指示灯（LED0，LED1，LED2）
    - 3个按键（KY0，KY1，KY2）
    - 1个TypeC-USB口，需下载USB程序
    - 1个SWD调试下载口
    - 1个复位按钮，可用于复位MCU
    - 1个电源开关，控制整个板的电源
    - 2个上拉5.1K电阻
    - 1个麦克风
    - 引出了20个GPIO口，均可独立外用

### QFN32开发板资源说明

资源说明部分，我们将分为两个部分说明：硬件资源说明和软件资源说明。

#### 硬件资源说明

这里我们首先详细介绍QFN32开发板的各个部分（图 1‑1中的标注部分）的硬件资源，我们将按逆时针的顺序依次介绍。

##### 引出IO口（总共有三处）

这是开发板IO引出端口，总共有三组IO引出口,其中包括2个芯片供电PAD口：VDD33、VDD42;1个复位口：RST，1个SWD调试口：CLK、DIO，20个GPIO口。

##### QFN32

这是开发板的核心芯片，型号为：QFN32。详细资料，请阅读：[sdk\doc\DataSheet](DataSheet/HYB53_58_DataSheet_CN_v1_3.docx)文档。

##### RF射频天线

这是开发板的2.4G天线，用于蓝牙无线射频。

##### 晶振16M

这是开发板外部晶振16MHz，9pF, 10ppm。

##### 3个按键

这是开发板板载的三个按键，在做程序调试，或者开发的时候可能需要按键来辅助，所以预留了三个，可以使用跳帽选择，如表 1‑1所示。

##### 3个LED灯

这是开发板板载的三个LED灯（LED0、LED1和LED2）。

QFN32开发板几乎每个实例都使用了LED来指示程序的运行状态，如表 1‑1所示。

表 1‑1 按键、LED选择

|  |  |  |  |
| --- | --- | --- | --- |
| 按键跳帽 | GPIO15 | GPIO16 | GPIO17 |
| KY0 | KY1 | KY2 |
| LED灯跳帽 | GPIO8 | GPIO9 | GPIO10 |
| LED0 | LED1 | LED2 |

##### 电源指示灯

这是开发板板载的一颗LED灯（PWR），用于指示电源状态。在电源开启的时候（通过板上的电源开关控制），该灯会亮，否则不亮。通过这个LED，可以判断开发板的上电情况。

##### 电源开关

这是开发板板载的电源开关。该开关用于控制整个开发板的供电，如果切断，则整个开发板都将断电，电源指示灯（PWR）会随着此开关的状态而亮灭。

##### USB

这是开发板模组与上位机串口通信的跳帽接口，如表 1‑2所示。

表 1‑2 USB/串口跳帽

|  |  |  |
| --- | --- | --- |
| QFN32芯片(USB) | GPIO6 | GPIO7 |
| 上位机 | U+ | U- |

##### 复位按钮

这是开发板板载的复位按键（RESET），用于复位QFN32，低电平有效。

##### SWD接口

这是开发板板载SWD调试接口，与JLINK或者STLINK等调试器（仿真器）连接。

#### 开发板IO引脚分配

表 1‑3 开发板IO 引脚分配

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| GPIO(独立) | 跳帽连接资源 | 连接关系说明 |  | GPIO(独立) | 跳帽连接资源 | 连接关系说明 |
| PA00 | SCK | SWD口 |  | PA11 | UP0 | 上拉5.1K电阻 |
| PA01 | SDI |  | PA12 | UP1 | 上拉5.1K电阻 |
| PA02 | MIB | 麦克风 |  | PA13 |  |  |
| PA03 | MIN |  | PA14 |  |  |
| PA04 |  |  |  | PA15 | KY0 | 按键0 |
| PA05 |  |  |  | PA16 | KY1 | 按键1 |
| PA06 | U+ | USB通信 |  | PA17 | KY2 | 按键2 |
| PA07 | U- |  | PA18 |  |  |
| PA08 | LED0 | LED0 |  | PA19 | RST | 复位按键 |
| PA09 | LED1 | LED1 |  |  |  |  |
| PA10 | LED2 | LED2 |  |  |  |  |

# 入门指南

本篇我们将详细介绍开发软件：MDK5。

通过该篇的学习，你将了解到：

* MDK5的安装
* 工程的编译；
* 软件仿真；
* 程序下载；
* 在线调试。

以上几个环节，概括了一个完整的B6x开发流程。本篇将图文并茂的向大家介绍这些内容，通过本篇的学习，希望大家能掌握B6x的开发流程，并能独立开始B6x的编程和学习。

本篇将分为如下3个章节：

1. 软件开发环境搭建；
2. 下载与调试；
3. 存储地址分配；

## 软件开发环境搭建

本章将向大家介绍MDK5软件的使用，通过本章的学习，我们最终将建立一个自己的MDK5工程，同时本章还将向大家介绍MDK5软件的一些使用技巧，希望大家在本章之后，能够对MDK5这个软件有个比较全面的了解。

本章分为如下几个小结：

### MDK5简介

MDK源自德国的KEIL公司，是RealView MDK的简称。在全球MDK被超过10万的嵌入式开发工程师使用。目前最新版本为：MDK5.25，该版本使用uVision5 IDE集成开发环境，是目前针对ARM处理器，尤其是Cortex M内核处理器的最佳开发工具。

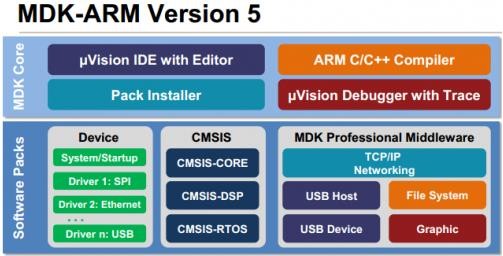


图 2‑1 MDK5组成

MDK5向后兼容MDK4和MDK3等，以前的项目同样可以在MDK5上进行开发(但是头文件方面得全部自己添加)，MDK5同时加强了针对Cortex-M微控制器开发的支持，并且对传统的开发模式和界面进行升级，MDK5由两个部分组成：MDK Core和Software Packs。其中，Software Packs可以独立于工具链进行新芯片支持和中间库的升级。如图 1‑1所示：

从上图可以看出，MDK Core又分成四个部分：uVision IDE with Editor（编辑器），ARMC/C++ Compiler（编译器），Pack Installer（包安装器），uVision Debugger with Trace（调试跟踪器）。uVision IDE从MDK4.7版本开始就加入了代码提示功能和语法动态检测等实用功能，相对于以往的IDE改进很大。

Software Packs（包安装器）又分为：Device（芯片支持），CMSIS（ARM Cortex微控制器软件接口标准）和Mdidleware（中间库）三个小部分，通过包安装器，我们可以安装最新的组件，从而支持新的器件、提供新的设备驱动库以及最新例程等，加速产品开发进度。

同以往的MDK不同，以往的MDK把所有组件到包含到了一个安装包里面，显得十分“笨重”，MDK5则不一样，MDK Core是一个独立的安装包，它并不包含器件支持、设备驱动、

CMSIS等组件，大小才300M左右，相对于MDK4.70A的500多M，瘦身明显，MDK5安装包可以在：<http://www.keil.com/demo/eval/arm.htm>下载到。而器件支持、设备驱动、CMSIS等组件，则可以点击MDK5的Build Toolbar的最后一个图标调出Pack Installer，来进行各种组件的安装。也可以在<http://www.keil.com/dd2/pack>这个地址下载，然后进行安装。

### MDK5安装

下载地址：[MDK\_KEIL](http://www.keil.com/fid/qtcbv3wb9c9j1wrdw6w1a24gf9liqqd1ig1yd1/files/umdkarm/MDK525.EXE)

1. keil版本需要安装5.25以上，或者直接使用我们链接提供的版本；
2. 根据提示完成软件的安装；
3. 将fromelf执行文件的所在路径添加系统环境变量中，重启keil生效，该文件所在路径在keil的安装目录下面Keil\_v5\ARM\ARMCC\bin，否则在使用keil编译时会报“fromelf不是内部或外部命令，也不是可运行的程序或批处理文件”的警告。（[如何设置添加环境变量](https://jingyan.baidu.com/article/47a29f24610740c0142399ea.html)）；

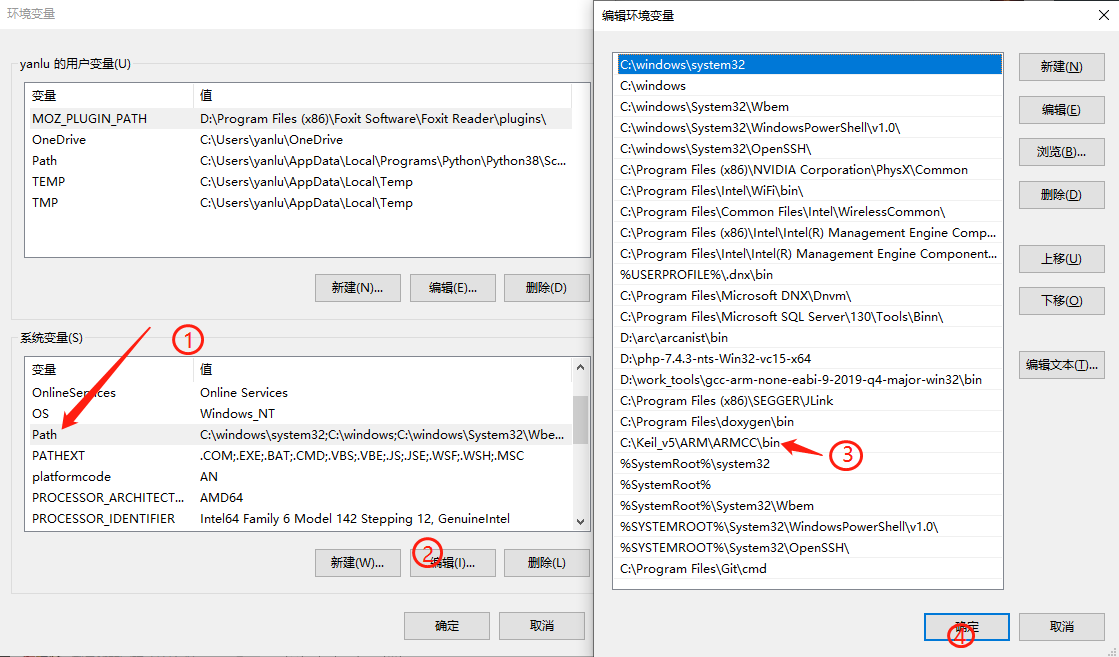


图 2‑2 添加系统变量

1. 在MDK5安装完成后，要让MDK5支持B6x的开发，还要安装B6x的器件支持包：Generic.B6x.1.0.0.pack（B6x的器件包）。这个包在开发板SDK\tools\KeilPack提供了，双击安装即可。

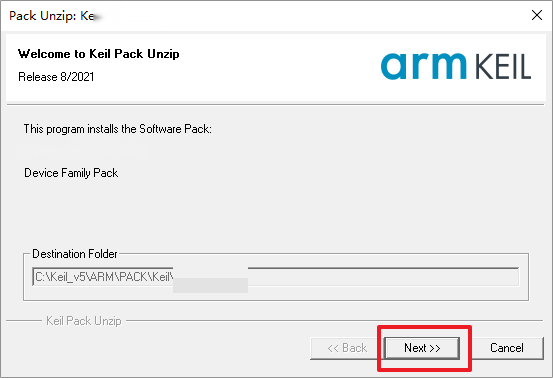
 

图 2‑3 Pack安装 图 2‑4 Pack安装成功

### Jlink驱动安装

1. 下载Jlink安装驱动

<https://www.segger.com/downloads/jlink/JLink_Windows.exe>

下载完成后，选择默认路径安装。

1. 将JLink安装路径JFlash.exe 、JLinkGDBServerCL.exe所在目录{JLINK\_SETUP\_DIR}添加到系统环境变量PATH。

### 分散加载文件浅释

分散加载（scatter）文件是一个文本文件，它可以用来描述连接器生成映像文件时需要的信息。通过编写一个分散加载文件来指定ARM连接器在生成映像文件时如何分配Code、RO-Data，RW-Data，ZI-Data等数据的存放地址。如果不用分散加载文件指定，那么ARM连接器会按照默认的方式来生成映像文件。一般情况下我们不需要使用分散加载文件。但对一些特殊的情况例如需要将不同的程序代码存储到不同的地址区域时需要修改分散加载文件。

1. 什么时候使用分散加载文件？

* 存在复杂的地址映射：例如代码和数据需要分开放在在多个区域
* 存在多种存储器类型：例如包含Flash，ROM，SDRAM等。我们根据代码与数据的特性把他们放在不同的存储器中，比如中断处理部分放在SRAM内部来提高响应速度，而把不常用到的代码放到速度比较慢的Flash内
* 函数的地址固定定位：可以利用Scatter file实现把某个函数放在固定地址，而不管其应用程序是否已经改变或重新编译
* 利用符号确定堆与堆栈
* 内存映射的IO：采用scatter file可以实现把某个数据段放在精确的地指处

1. ARM程序的组成

要了解分散加载文件前，首先需要对ARM程序中的各个部分进行了解，可参考图 2‑5：

* Code：为程序代码部分
* RO-Data：表示程序定义的常量及 const 型数据
* RW-Data：表示已经初始化的静态变量，变量有初值
* ZI-Data：表示未初始化的静态变量，变量无初值



图 2‑5 各类型数据声明

Keil工程编译完成后，查看其map文件，可以看到如图 2‑6的信息。

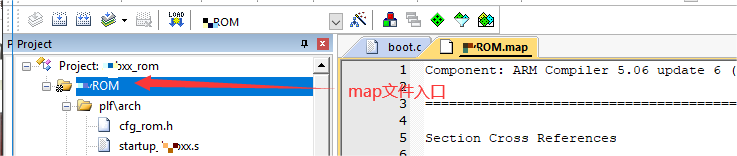


图 2‑6 keil工程中map文件入口

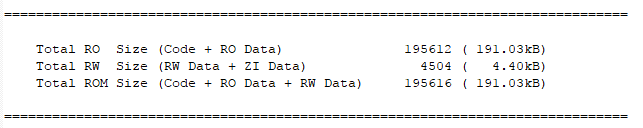


图 2‑7 map文件信息

由图 1‑7可以的map文件可以看出：

ROM（Flash）Size = Code＋RO-Data＋RW-Data =191.03KB

RAM Size = RW-Data＋ZI-Data = 4.40KB

为什么上述的RW-Data既占用Flash又占用RAM么，变量不是放在RAM中的么，为什么会占用 Flash？因为RW数据不能像ZI那样“无中生有”的，ZI数据只要求其所在的区域全部初始化为零，所以需要程序根据编译器给出的ZI基地址及大小来将相应的RAM清零。但RW数据却不这样做，所以编译器为了完成所有RW数据赋值，其先将RW段的所有初值，先保存到Flash中，程序执行时，再Flash中的数据搬运到RAM中，所以RW段即占用Flash又占用RAM，且占用的空间大小是相等的。

1. 分散加载文件语法

分散加载文件主要由一个加载时域和多个运行时域组成，大致结构如图 2‑8所示：

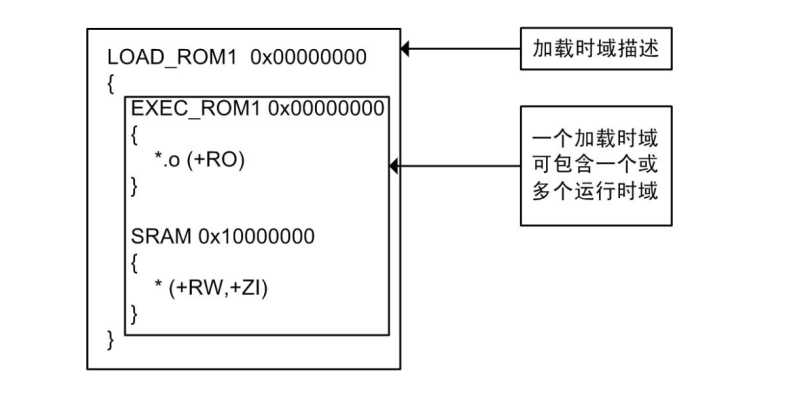


图 2‑8 分散加载文件语法结构

1. 加载时域描述

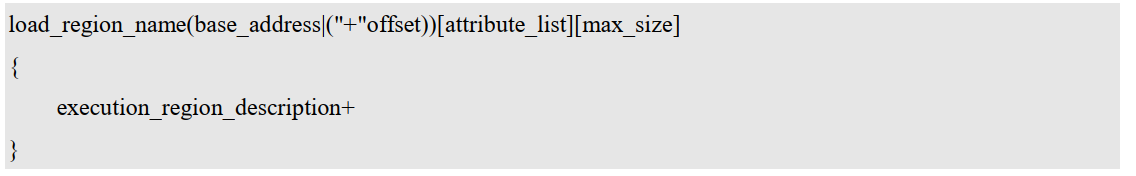


图 2‑9 加载时域语法结构

* load\_region\_name：为本加载时域的名称，名称可以按照用户意愿自己定义，该名称中只有前31个字符有意义
* base\_designator：用来表示本加载时域的起始地址，可以有下面两种格式中的一种；
  + base\_address：表示本加载时域中的对象在连接时的起始地址，地址必须是字对齐的
  + +offset：表示本加载时域中的对象在连接时的起始地址是在前一个加载时域的结束地址后偏移量offset字节处。本加载时域是第一个加载时域，则它的起始地址即为offset，且offset必须能被4整除。
* attribute\_list：指定本加载时域内容的属性，包含以下几种，默认加载时域的属性是ABSOLUTE。
* ABSOLUTE：绝对地址；
* PI：与位置无关；
* RELOC：可重定位；
* OVERLAY：覆盖；
* NOCOMPRESS：不能进行压缩。
* max\_size：指定本加载时域的最大尺寸。如果本加载时域的实际尺寸超过了该值，连接器将报告错误，默认取值为 0xFFFFFFFF；
* execution\_region\_description：表示运行时域，后面有个+号，表示其可以有一个或者多个运行时域，关于运行时域的介绍请看后面。

1. 运行时域描述

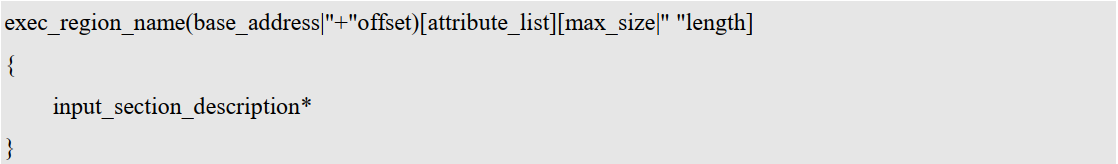


图 2‑10 运行时域语法结构

* exec\_region\_name：为本加载时域的名称，名称可以按照用户意愿自己定义，该名称中只有前31个字符有意义；
* base\_address：用来表示本加载时域的起始地址，可以有下面两种格式中的一种：
* base\_address：表示本加载时域中的对象在连接时的起始地址，地址必须是字对齐的；
* +offset：表示本加载时域中的对象在连接时的起始地址是在前一个加载时域的结束地址后偏移量offset字节处，offset的值必须能被4整除。
* attribute\_list：指定本加载时域内容的属性：
* ABSOLUTE：绝对地址；
* PI：与位置无关；
* RELOC：可重定位；
* OVERLAY：覆盖；
* FIXED：固定地址。加载地址和执行地址都是由基址指示符指定的，基址指示符必须是绝对基址，或者偏移为0。
* ALIGNalignment：将执行区的 对齐约束从4增加到alignment。 alignment必须为2的正数幂。如果执行区具有base\_address，则它必须为alignment对齐。如果执行区具有offset，则链接器将计算的区基址与alignment边界对齐；
* EMPTY：在执行区中保留一个给定长度的空白内存块，通常供堆或堆栈使用。
* ZEROPAD：零初始化的段作为零填充块写入ELF文件，因此，运行时无需使用零进行填充；
* PADVALUE：定义任何填充的值。如果指定PADVALUE，则必须为其赋值；
* NOCOMPRESS：不能进行压缩；
* UNINIT：未初始化的数据。
* max\_size：指定本加载时域的最大尺寸。如果本加载时域的实际尺寸超过了该值，连接器将报告错误，默认取值为0xFFFFFFFF；
* length：如果指定的长度为负值，则将base\_address作为结束地址。它通常与EMPTY一起使用，以表示在内存中变小的堆栈。
* input\_section\_description：指定输入段的内容。

1. 输入段描述

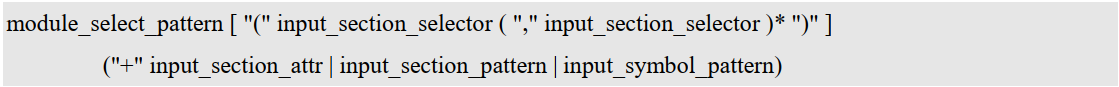


图 2‑11 输入段语法结构

* module\_select\_pattern：目标文件滤波器，支持使用通配符“\*”与“?”。其中符号“ \*”代表零个或多个字符，符号“？”代表单个字符。进行匹配时所有字符不区分大小写。当 module\_select\_pattern与以下内容之一相匹配时，输入段将与模块选择器模式相匹配：
* 包含段和目标文件的名称；
* 库成员名称（不带前导路径名）；
* 库的完整名称（包括路径名）。如果名称包含空格，则可以使用通配符简化搜索。例如，使用\*libname.lib 匹配C:\lib\_dir\libname.lib。
* input\_section\_attr：属性选择器与输入段属性相匹配。每个input\_section\_attr的前面有一个“+”号。如果指定一个模式以匹配输入段名称，名称前面必须有一个“+”号。可以省略紧靠“+”号前面的任何逗号。选择器不区分大小写。可以识别以下选择器：
* RO-CODE；
* RO-DATA；
* RO，同时选择 RO-CODE 和 RO-DATA；
* RW-DATA；
* RW-CODE；
* RW，同时选择 RW-CODE 和 RW-DATA；
* ZI；
* ENTRY：即包含 ENTRY 点的段。  
  可以识别以下同义词：
* CODE 表示 RO-CODE；
* CONST 表示 RO-DATA；
* TEXT 表示 RO；
* DATA 表示 RW；
* BSS 表示 ZI。  
  可以识别以下伪属性：
* FIRST；
* LAST。  
  通过使用特殊模块选择器模式 .ANY可以将输入段分配给执行区，而无需考虑其父模块。可以使用一个或多个.ANY 模式以任意分配方式填充运行时域。在大多数情况下，使用单个.ANY等效于使用\*模块选择器。

### 分散加载文件使用说明

B系列芯片支持两种Code方式：1.Flash Code；2.Sram Code

Flash Code：将Code通过UART或者Jlink等方式烧录到Flash中，Code在Flash片上执行。

Sram Code：将Code烧录到Flash和其他存储设备中，或者通过Jlink直接加载到Sram上，使用时需要将Code拷贝到对应的Sram地址上执行。

根据应用需求将Code分地址储存和执行，可以通过分散加载机制实现，分散加载通过配置分散加载文件来完成应用需求。在SDK\core\sct 中有四个可供选择的分散加载文件(\*.sct)文件，用来修改Code、RO-Data，RW-Data，ZI-Data等数据的存放地址。以SDK\examples\gpioTest工程为例

使用方法为：**Project -> Options for Target -> Linker** 中

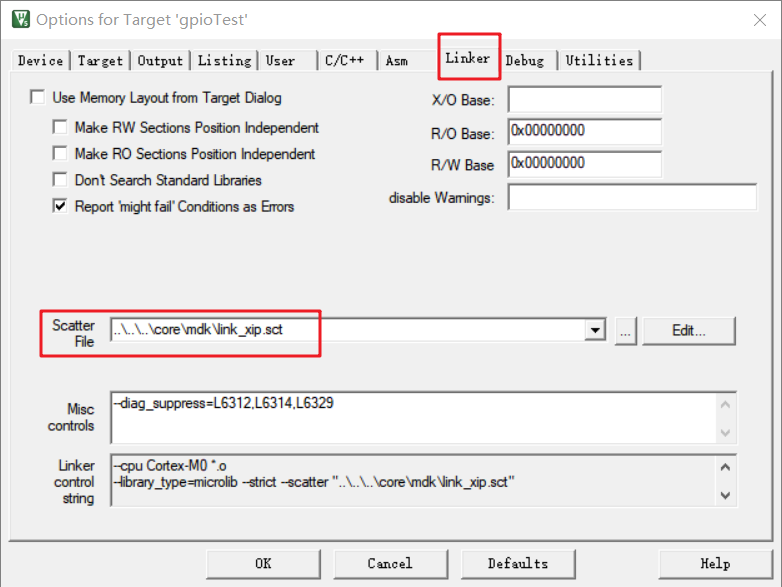


图 2‑12 分散加载文件位置

## 下载与调试

本章，我们将向读者介绍B6x的代码下载以及调试。这里的调即硬件调试（在线调试），必须有仿真器（JLINK/ULINK/STLINK等）才可以。通过本章的学习，你将了解到：

* B6x的程序下载；
* 利用JLINK对B6x进行在线调试；

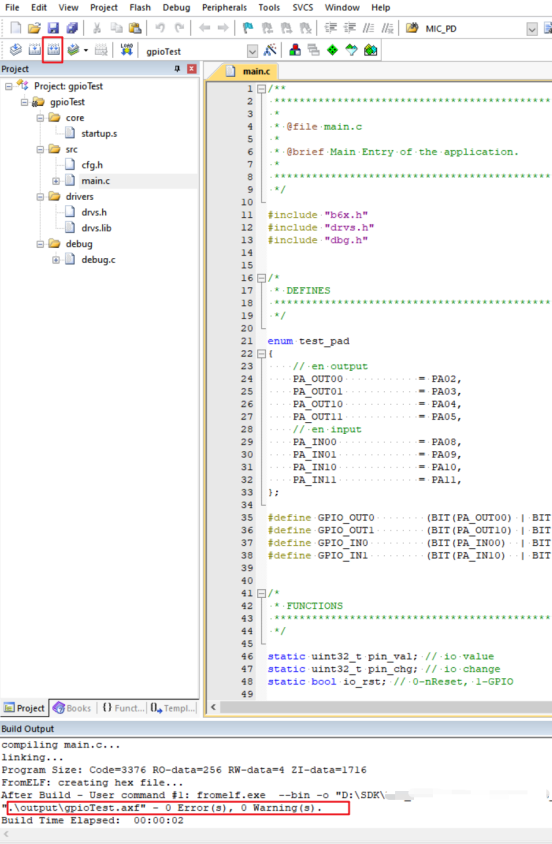
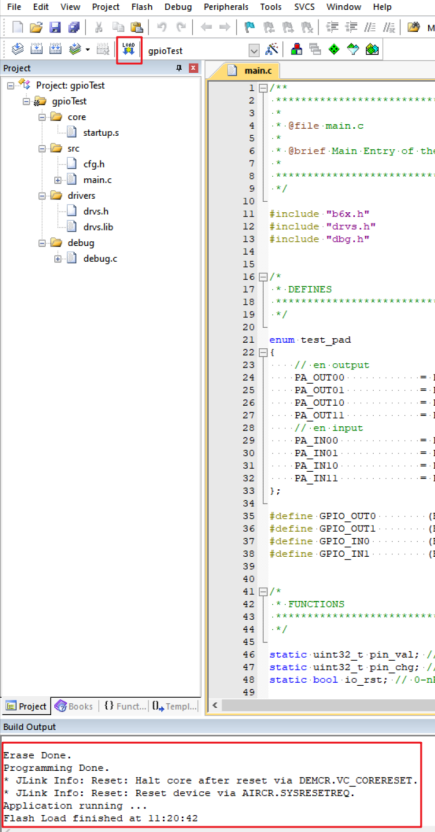
本章将分为如下2个小节：

### 程序下载

#### SWD下载

1. 编译工程文件

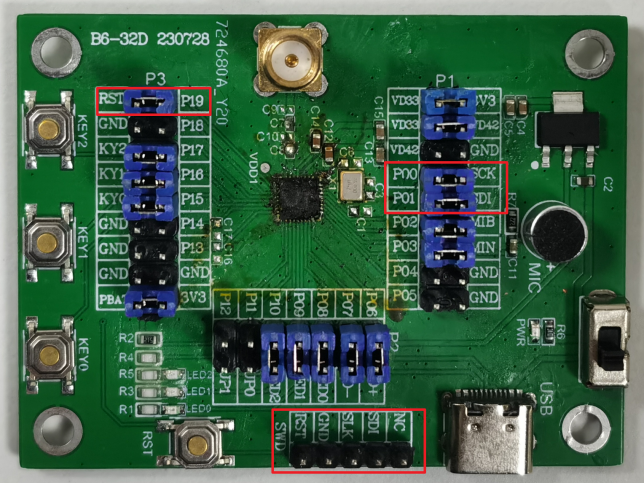
打开工程编译前, 确保SDK\tools\KeilPack\Generic.B6x.1.0.0.pack已被安装, 编译没有报错后,可以进行下载.

1. 下载程序

使用Jlink的SWD口, 连接开发板的SWD口,如下图. 注意:

P00跳冒短接SCK,P01跳冒短接SDI,P19跳冒短接RST.



#### Boot Uart下载

1. 进入Boot模式方式1
2. 上电后, 同时拉低Uart引脚: Tx(PA06)/Rx(PA07)和RST复位引脚;
3. 15ms后, 先恢复RST复位引脚;
4. 20ms后, 再恢复Uart引脚;
5. Boot Uart发送”B9”, 需要回复”9B”.(波特率115200,停止位1,数据位:8,校验位:无)
6. 100ms后, 未收到回复, 退出Boot模式.
7. 进入Boot模式方式2
8. 上电前, 拉低Uart引脚: Tx(PA06)/Rx(PA07); (**确保掉电干净**)
9. 上电20ms后, 恢复Uart引脚;
10. Boot Uart发送”B9”, 需要回复”9B”.(波特率115200,停止位1,数据位:8,校验位:无)
11. 100ms后, 未收到回复, 退出Boot模式.

Boot进入后测试指令: 5A 2A 00 00

响应指令: A5 2A 01 00 A3

### 在线调试

上一节，我们介绍了如何通过利用Keil给B6x下载代码，并在B6x开发板上验证了我们程序的正确性。这个代码比较简单，所以不需要硬件调试，我们直接就一次成功了。可是，如果你的代码工程比较大，难免存在一些bug，这时，就有必要通过在线调试来解决问题了。

串口只能下载代码，并不能实时跟踪调试，而利用调试工具，比如JLINK、ULINK、STLINK等就可以实时跟踪程序，从而找到你程序中的bug，使你的开发事半功倍。这里我们以JLINK为例，说说如何在线调试B6x。

JLINK需要支持SWD功能（购买需确认），SWD调试的时候占用的IO线很少，只需要两根即可。

JLINK的驱动安装比较简单，我们在这里就不说了。在安装了JLINK的驱动之后，我们接上JLINK，并把SWD口插到B6x开发板上，打开一个工程，点击，打开Options for Target 选项卡，在Debug栏选择仿真工具为J-LINK/J-TRACE Cortex，如图 2‑13所示：

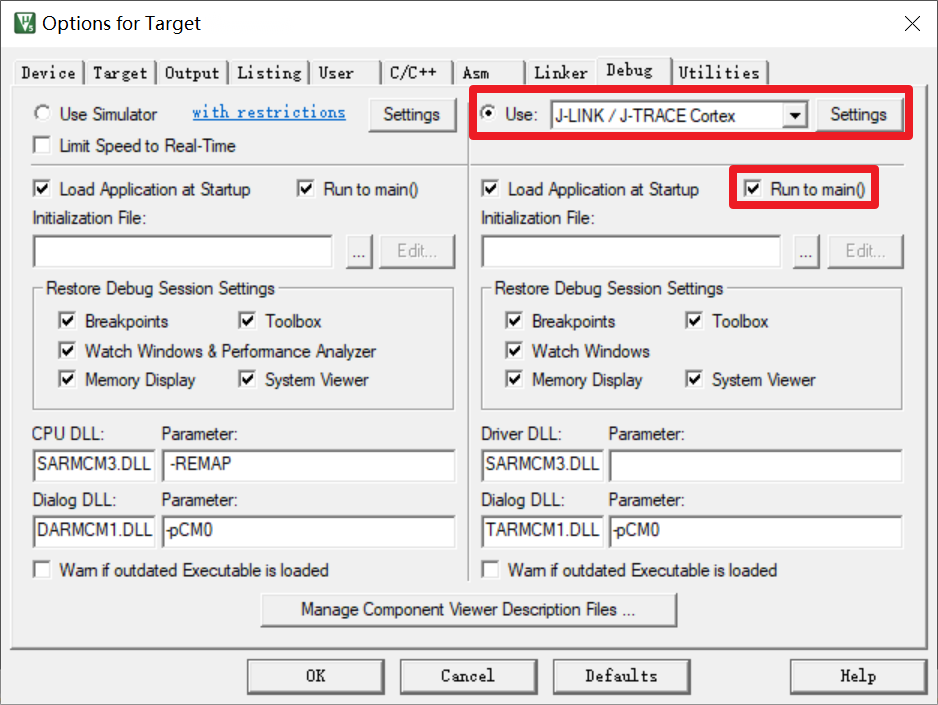


图 2‑13 Debsug选项卡设置

图 1‑29中我们还勾选了Run to main()，该选项选中后，只要点击仿真就会直接运行到main函数，如果没选择这个选项，则会先执行startup.s文件的Reset\_Handler，再跳到main函数。

然后我们点击Settings，设置J-LINK的一些参数，如图 2‑14所示：

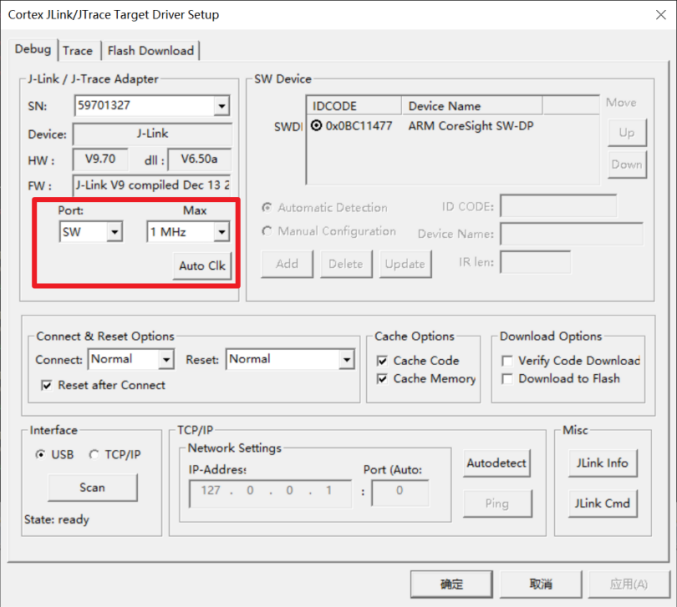


图 2‑14 J-LINK模式设置

图 1‑30中，我们使用J-LINK V9的SW模式调试，我们建议大家在调试的时候，一定要选择SW模式。Max Clock，可以点击Auto Clk来自动设置，图中我们设置SWD的调试速度为1Mhz，这里，如果你的USB数据线比较差，那么可能会出问题，此时，你可以通过降低这里的速率来试试。

单击OK，完成此部分设置，接下来我们还需要在Utilities选项卡里面设置下载时的目标编程器，如图 2‑15所示：

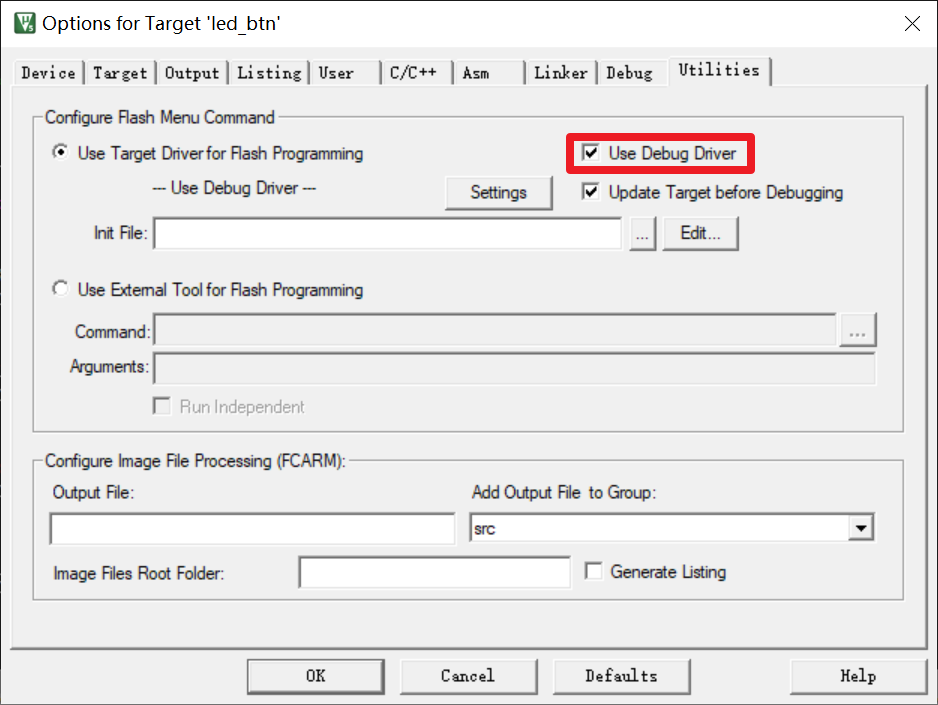


图 2‑15 FLASH编程器选择

图 1‑31中，我们直接勾选Use Debug Driver，即和调试一样，选择JLINK来给目标器件的FLASH编程，然后点击Settings，设置如图 2‑16所示：

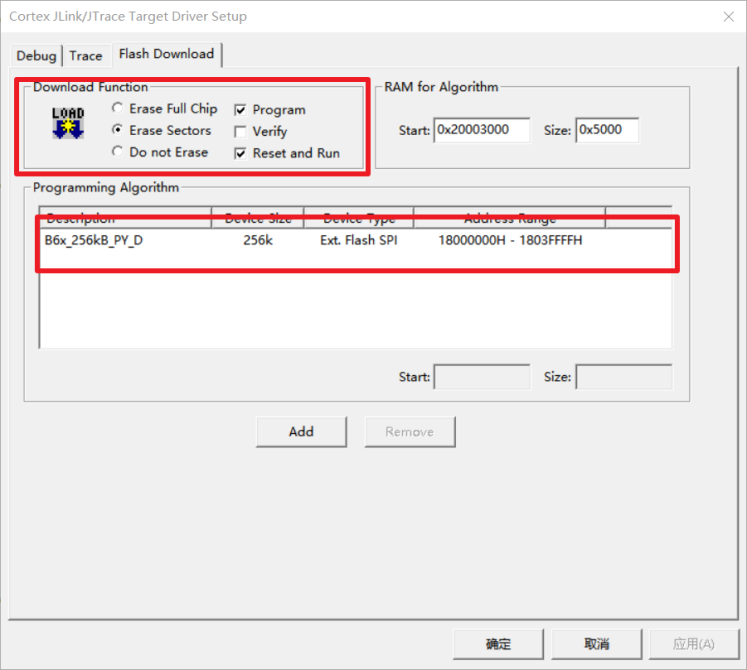


图 2‑16 编程设置

这里MDK5会根据我们新建工程时选择的目标器件，自动设置flash算法。我们使用的是B6x，FLASH容量为256K字节，所以Programming Algorithm里面默认会有256K型号的B6x FLASH算法。最后，选中Reset and Run选项，以实现在编程后自动运行，其他默认设置即可。

选择适当的分散加载文件，在SDK\core\sct下，有四个分散加载文件，具体作用请看上一章，B6x芯片一般选择link\_xip.sct。

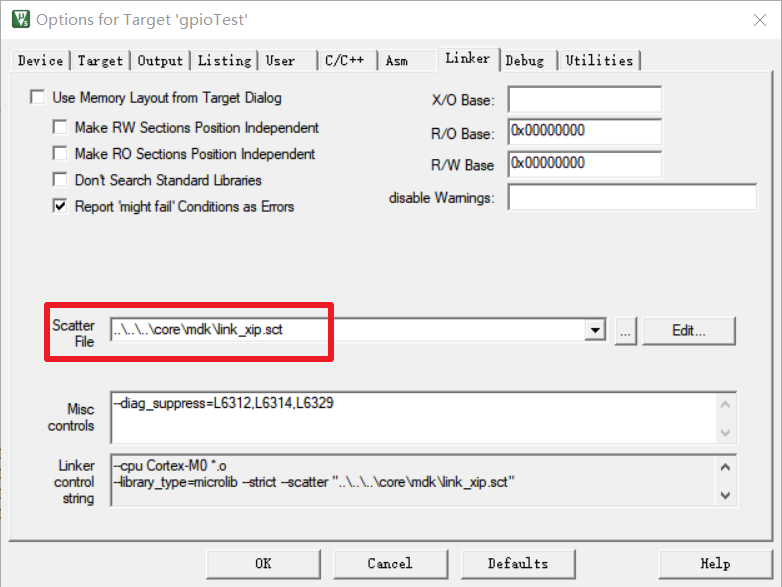


图 2‑17 分散加载文件选择

在设置完之后，点击确认，然后再点击OK，回到IDE界面，编译一下工程。然后点击（开始/停止仿真按钮），开始仿真（如果开发板的代码没被更新过，则会先更新代码（即下载代码），再仿真，你也可以通过按，只下载代码，而不进入仿真。），如图 2‑18所示：

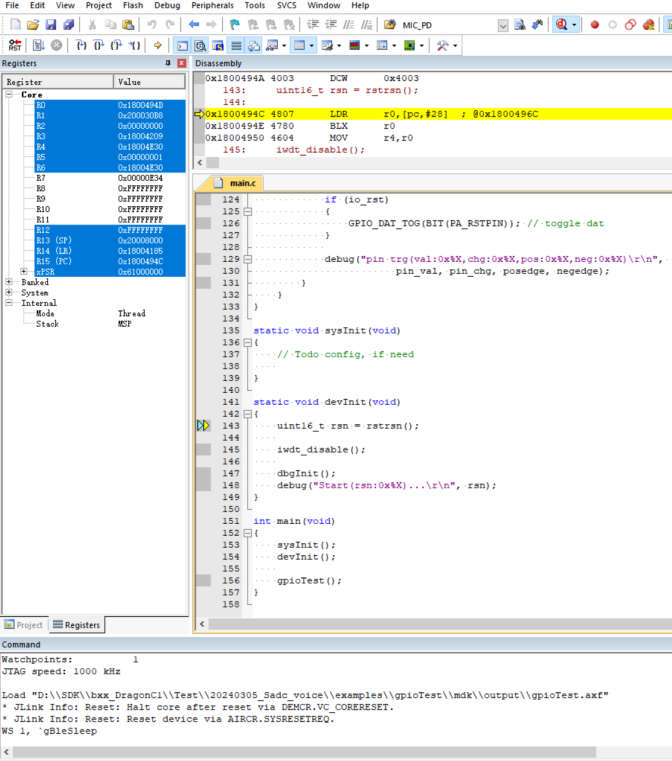


图 2‑18 开始仿真

因为我们之前勾选了Run to main()选项，所以，程序直接就运行到了main函数的入口处。另外，此时MDK多出了一个工具条，这就是Debug工具条，这个工具条在我们仿真的时候是非常有用的，下面简单介绍一下Debug工具条相关按钮的功能。Debug工具条部分按钮的功能如图 2‑19所示: 

复位

执行到断点处

停止执行

执行进去

执行过去

执行出去

执行到光标处

汇编窗口

堆栈局部变量窗口

观察窗口

内存查看窗口

内存打印窗口

逻辑分析窗口

系统查看窗口

图 2‑19 Debug工具条

复位：其功能等同于硬件上按复位按钮。相当于实现了一次硬复位。按下该按钮之后，代码会重新从头开始执行。

执行到断点处：该按钮用来快速执行到断点处，有时候你并不需要观看每步是怎么执行的， 而是想快速的执行到程序的某个地方看结果，这个按钮就可以实现这样的功能，前提是你在查看的地方设置了断点。

停止运行：此按钮在程序一直执行的时候会变为有效，通过按该按钮，就可以使程序停止下来，进入到单步调试状态。

执行进去：该按钮用来实现执行到某个函数里面去的功能，在没有函数的情况下，是等同于执行过去按钮的。

执行过去：在碰到有函数的地方，通过该按钮就可以单步执行过这个函数，而不进入这个函数单步执行。

执行出去：该按钮是在进入了函数单步调试的时候，有时候你可能不必再执行该函数的剩余部分了，通过该按钮就直接一步执行完函数余下的部分，并跳出函数，回到函数被调用的位置。

执行到光标处：该按钮可以迅速的使程序运行到光标处，其实是挺像执行到断点处按钮功能，但是两者是有区别的，断点可以有多个，但是光标所在处只有一个。

汇编窗口：通过该按钮，就可以查看汇编代码，这对分析程序很有用。

堆栈局部变量窗口：通过该按钮，显示Call Stack+Locals窗口，显示当前函数的局部变量及其值，方便查看。

观察窗口：MDK5提供2个观察窗口（下拉选择），该按钮按下，会弹出一个显示变量的窗口，输入你所想要观察的变量/表达式，即可查看其值，是很常用的一个调试窗口。

内存查看窗口：MDK5提供4个内存查看窗口（下拉选择），该按钮按下，会弹出一个内存查看窗口，可以在里面输入你要查看的内存地址，然后观察这一片内存的变化情况。是很常用的一个调试窗口

串口打印窗口：MDK5提供4个串口打印窗口（下拉选择），该按钮按下，会弹出一个类似串口调试助手界面的窗口，用来显示从串口打印出来的内容。

逻辑分析窗口：该图标下面有3个选项（下拉选择），我们一般用第一个，也就是逻辑分析窗口(Logic Analyzer)，点击即可调出该窗口，通过SETUP按钮新建一些IO口，就可以观察这些IO口的电平变化情况，以多种形式显示出来，比较直观。

系统查看窗口：该按钮可以提供各种外设寄存器的查看窗口（通过下拉选择），选择对应外设，即可调出该外设的相关寄存器表，并显示这些寄存器的值，方便查看设置的是否正确。

Debug工具条上的其他几个按钮用的比较少，我们这里就不介绍了。以上介绍的是比较常用的，当然也不是每次都用得着这么多，具体看你程序调试的时候有没有必要观看这些东西，来决定要不要看。

# 常见问题

### 找不到USB设备?

H:这个问题可能有几个原因:

1. 未下载USB, 驱动程序。
2. U-/U+ 跳冒未接。

### JLINK 能否给开发板供电？

H：不能，也不建议。因为开发板功耗比较大，JLINK 如果又给自己供电又给开发板供电，会很容易导致 JLINK 固件丢失。

### JLINK第一次烧录后无法再次烧录？

H:排除硬件问题后。如果Flash中下载的程序出现问题，导致Jlink无法接管芯片，可以将PA6和PA7接地，然后按复位键，让芯片进入BOOT-Uart模式，可再次烧录。